



Universidade Federal
de São João del-Rei

Documentação Trabalho Prático 2

Gabriel Souza de Oliveira e Ian Carvalho Ramos
May 21, 2023

Contents

1	Introdução	2
2	Implementação	3
2.1	Funções Auxiliares	3
2.2	Estratégia 1 - Programação Dinâmica	3
2.2.1	Apresentação da solução	3
2.2.2	Listagem das rotinas, variáveis e explicação geral . . .	3
2.2.3	Explicação e aplicação das rotinas	5
2.2.4	Análise de complexidade	5
2.3	Estratégia 2 - Recursivo	5
2.3.1	Apresentação da solução	5
2.3.2	Listagem das rotinas e explicação geral	6
2.3.3	Explicação e aplicação das rotinas	6
2.3.4	Análise de complexidade	7
3	Resultados e Discussões	7
3.1	Apresentação dos casos de teste	7
3.2	Resultados - Solução Dinâmica	8
3.3	Resultados - Solução Recursiva	10
3.4	Comparação entre os 2 algoritmos	11
4	Conclusão	12

1 Introdução

O código foi criado para ajudar Harry a vencer um jogo, em que ele precisa escolher um caminho cuja sua energia inicial seja mínima para chegar no final. Esse problema é representado por uma matriz, em que a posição (1, 1) é a posição inicial de Harry, e a posição (R, C), contendo o artefato mágico, é a posição em que ele deve chegar, sendo R o número de linhas da matriz e C o número de colunas. Cada elemento dessa matriz pode conter uma poção de energia ou um monstro, que Harry deve enfrentar. As poções estão nos elementos com valores positivos, e seu valor é acrescentado na energia atual de Harry, já os monstros são os elementos com valores negativos, em que Harry perde esse valor em energia para enfrentar o monstro e chegar em tal posição. A solução do problema, portanto, é a menor energia possível com que Harry deve iniciar na primeira posição (1, 1) para que ele chegue na última posição (R, C), podendo andar somente para a direita ou para baixo.

O problema de encontrar a energia mínima necessária para que Harry Potter alcance o artefato mágico se assemelha bastante a problemas de otimização de caminhos em um grafo, em que o peso das arestas é a energia que ele ganha ou perde ao chegar no vértice, e representa um elemento da matriz, e é somado na energia atual de Harry, podendo ser positivos ou negativos. Entretanto, o problema não consiste em encontrar um caminho com o menor gasto de energia, e sim em determinar uma energia inicial mínima para que nenhuma aresta de peso negativo deixasse a energia menor ou igual a 0 ao longo do caminho.

Foram descritas a execução e os resultados de duas soluções diferentes para o problema nesta documentação, que está organizada da seguinte forma: Na seção "Implementação", é feita a apresentação das funções auxiliares, a primeira estratégia e a segunda estratégia utilizadas para solucionar o problema. A próxima seção se refere aos "Resultados e Discussões" onde são expostos os resultados de alguns testes realizados para verificar o funcionamento do programa para cada estratégia, seguido de suas respectivas complexidades de tempo e espaço. Por fim, a seção "Conclusão", onde concluímos a apresentação do trabalho prático.

2 Implementação

2.1 Funções Auxiliares

A princípio, dentro da *main* de cada estratégia é utilizada uma função fundamental para a execução do código. Essa função é responsável por ler os dados de um arquivo de texto. Esse método inicia fazendo a leitura da primeira linha do texto de entrada, que contém o número de casos de teste, que fica armazenado em uma variável inteira chamada "t". A partir disso, baseado no número de casos de teste, o código realiza um *loop* que itera t vezes, e para cada um ele armazena nas variáveis "r" e "c" o número de linhas e de colunas da matriz que está sendo analisada, respectivamente. A partir dessas duas variáveis são aplicadas outras funções e criadas outras variáveis para solucionar o problema de duas maneiras diferentes, que serão apresentadas em suas respectivas seções.

Além disso, foi criada uma função para liberar a memória alocada para uma matriz, que será utilizada no final dos códigos.

2.2 Estratégia 1 - Programação Dinâmica

2.2.1 Apresentação da solução

A primeira estratégia utiliza a programação dinâmica para preencher a matriz "matriz_energia_min" em que a posição (I, J) indica a energia mínima inicial para que Harry consiga chegar nessa posição sem que sua energia fique menor ou igual a 0, e cada posição dessa matriz é utilizada para preencher as posições que se encontram à sua direita ou em baixo dela. Para que esse método fosse utilizado com êxito, foram utilizadas 3 matrizes, a matriz "matriz_principal", que armazena a matriz contida no arquivo de entrada, a matriz "m_saldo_energia", que armazena a energia que Harry possui ao chegar em cada posição pela casa anterior com a menor energia inicial, a matriz "matriz_energia_min" anteriormente apresentada, que armazena a energia mínima para chegar em cada posição.

2.2.2 Listagem das rotinas, variáveis e explicação geral

Para entender a aplicação das funções utilizadas para preencher a matriz de energia mínima, primeiro é necessário entender como ela é calculada. Esse cálculo é feito utilizando a energia mínima da posição de cima e da esquerda da posição a ser calculada. Após realizar uma comparação das

duas, é selecionada a posição com o menor valor, e então pegamos o saldo de energia dessa posição e acrescentamos o valor contido na matriz principal da posição a ser calculada. Se o saldo continuar maior ou igual a 1, então a posição na matriz de energia mínima recebe o valor de energia mínima do vizinho selecionado, e o saldo de energia dessa posição recebe o saldo da posição anterior somado com o valor dessa posição na matriz principal. Já se o saldo ficar menor ou igual a 0, então o saldo da posição analisada se torna 1, e a energia mínima se torna a energia mínima do vizinho selecionado somado com o valor do saldo dele subtraído do módulo da posição analisada na matriz principal e somado com 1. Para preencher a primeira linha e a primeira coluna da matriz, o mesmo método é utilizado, porém sem a parte de comparar os vizinhos, já que as posições só possuem 1 possível vizinho do qual Harry pode chegar.

Em relação às variáveis, foram utilizadas três matrizes diferentes, uma para armazenar a matriz contida no arquivo de entrada, uma para armazenar o valor mínimo da casa inicial para chegar em cada posição, e uma para armazenar a energia que Harry tem guardada em cada posição. No final o valor da energia mínima inicial é armazenada em uma variável e imprimida no arquivo de saída.

As funções utilizadas para a solução do problema foram:

`preencher_bordas_memin(matriz principal, matriz de energia mínima, matriz de saldo de energia, número de linhas, número de colunas)`: preenche a primeira linha e a primeira coluna da matriz de energia mínima.

`menor_energia_inicial(matriz principal, matriz de energia mínima, matriz de saldo de energia, número da linha, número da coluna)`: retorna o valor de energia mínima para a posição da matriz referente ao número da linha e da coluna.

`preencher_matriz_energia_min(matriz principal, matriz de energia mínima, matriz de saldo de energia, número de linhas, número de colunas)`: contém dois *loops* encadeados para preencher a matriz de energia mínima utilizando a função "`menor_energia_inicial`" para cada posição, com exceção da primeira linha e da primeira coluna.

`menor_energia_inicial_matriz(matriz de energia mínima, número de linhas, número de colunas)`: retorna o valor de energia mínima da primeira casa para que a última casa seja alcançada sem que a energia fique menor ou igual a 0.

2.2.3 Explicação e aplicação das rotinas

Para encontrar a energia mínima da primeira casa, primeiro consideramos ela como "1", pois é o menor valor inteiro que não é menor ou igual a zero. A partir dela, optamos por criar uma função "preencher_bordas_memin", que recebe as três matrizes apresentadas, o número de linhas e colunas, e preenche as bordas de cima e da esquerda da matriz de energia mínima e da matriz de saldo de energia, ou seja, a primeira linha e a primeira coluna, pois só existe um caminho possível para alcançar suas posições, que são andando somente para a direita ou andando somente para baixo, respectivamente. Com essa função sendo aplicada na matriz principal, é possível calcular a energia mínima para as posições restantes da matriz, pois após preenchidas as bordas, ao preencher linha por linha, todas as posições já irão possuir a energia mínima calculada de seus "vizinhos" de cima e da esquerda.

Em relação às posições que não pertencem à primeira linha ou à primeira coluna, foi utilizada a função "menor_energia_inicial", que recebe os mesmos parâmetros da função utilizada para preencher as bordas e preenche o resto da matriz de energia mínima, incluindo a última casa, que é a solução do problema. Essa função é utilizada na função "preencher_matriz_energia_min", que utiliza dois *loops* encadeados, cujos valores iniciais são 1, já que começam na segunda posição de cada linha e coluna e vão até o número de linhas e colunas menos um, e para cada posição é calculada sua energia mínima. Já para extrair o valor da última casa já calculado, é utilizada a função "menor_energia_inicial_matriz", que recebe a matriz de energia mínima, o número de linhas e o número de colunas, e retorna o valor da última casa.

2.2.4 Análise de complexidade

Como são realizadas um número fixo de comparações para preencher cada posição da matriz de energia mínima, que possui tamanho $m * n$, sendo m o número de linhas e n o número de colunas, então a complexidade dessa solução é $O(m * n)$, já que qualquer constante maior que o número fixo de comparações multiplicando $(m * n)$ dominará assintoticamente a função correspondente à complexidade desse algoritmo.

2.3 Estratégia 2 - Recursivo

2.3.1 Apresentação da solução

A segunda estratégia utiliza uma abordagem recursiva para percorrer a matriz e encontrar a quantidade de vida mínima inicial necessária para alcançar

o artefato na posição (R, C) da matriz.

2.3.2 Listagem das rotinas e explicação geral

A princípio, foram criadas duas funções auxiliares: `int min(int a, int b)` e `int max(int a, int b)`. A função `int max` recebe dois valores inteiros e compara esses dois valores. Se o valor de "a" for maior que o valor de "b", retorna o valor de "a", caso contrário, retorna o valor de "b". Analogamente, a função `int min` compara os valores de "a" e "b" e, se o valor de "a" for menor que "b", retorna "a", caso contrário, retorna "b". Ademais, fornecida pela biblioteca "stdlib.h", a função "abs" é utilizada. Essa função é responsável retornar o valor absoluto ou positivo de um inteiro.

A ideia por trás do algoritmo recursivo é que, para cada posição da matriz, é preciso encontrar o caminho que exige energia mínima para alcançar o artefato mágico na última posição a partir daquela célula. Isso é feito por meio de uma verificação entre os dois caminhos possíveis a partir da posição atual: para baixo (i+1, j) ou para a direita (i, j+1). Essa abordagem recursiva é uma solução viável para o problema, porém pode levar a uma complexidade de tempo exponencial no pior caso, já que algumas posições do grid podem ser visitadas várias vezes. Em casos de matrizes grandes, é necessário utilizar uma outra abordagem mais eficiente, como a solução por programação dinâmica, que foi abordada e explicada na estratégia anterior.

2.3.3 Explicação e aplicação das rotinas

A função `int solucao1` recebe como parâmetro: a matriz, o número de linhas, o número de colunas, e duas variáveis inteiras "i" e "j". Começando na célula (i,j), a recursão verifica se este é o último local (linhas-1, colunas-1). Se for esse o caso, a função retorna o maior valor entre 1 e a energia atual de Harry; ou seja, se Harry produzir energia positiva após chegar à última célula, ele só precisa de mais 1 energia para chegar ao escapamento. Caso contrário, a função retorna a diferença entre a energia atual de Harry e a energia necessária para chegar à última célula + 1, ou seja, Harry precisa da energia necessária para chegar lá mais uma energia adicional de 1 para evitar morrer no caminho.

A função determina se a posição está dentro dos limites do grid e retorna o valor mais alto se a célula atual não for a célula final. Caso contrário, a função chama a si mesma para procurar possíveis rotas que comecem na célula atual. A energia mínima necessária é então definida como o máximo

entre a energia atual de Harry menos a energia da célula atual e a energia necessária para seguir o caminho de energia mínima da célula atual. Para garantir que Harry não morra na estrada, o valor retornado é o menor desses dois números mais 1.

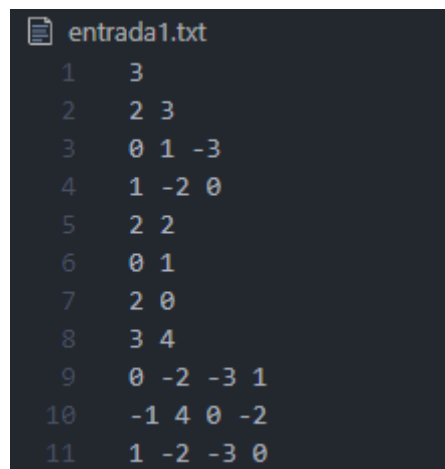
2.3.4 Análise de complexidade

A complexidade de tempo da estratégia recursiva é $O(2^{M+N})$ onde M é o número de linhas e N é o número de colunas da matriz, portanto, ele é ineficiente conforme o número de linhas e colunas da matriz cresce, tornando-o viável apenas para matrizes pequenas.

3 Resultados e Discussões

3.1 Apresentação dos casos de teste

Primeiro caso:



```
entrada1.txt
1 3
2 2 3
3 0 1 -3
4 1 -2 0
5 2 2
6 0 1
7 2 0
8 3 4
9 0 -2 -3 1
10 -1 4 0 -2
11 1 -2 -3 0
```

Figure 1: arquivo de entrada do primeiro caso de teste

Segundo caso:

Terceiro caso: 2 matrizes com valores aleatórios de -100 até 100, a primeira com dimensões iguais a 20x20 e a segunda com dimensões iguais a 250x250. Quarto caso: matriz com valores aleatórios de -100 até 100 com dimensão igual a 2500x2500.


```

entrada2.txt
1 1
2 14 14
3 0 81 840 216 -879 -149 -825 932 850 -46 -565 454 549 39
4 340 542 -405 392 -704 -176 -854 -556 662 -385 -268 -979 -286 -934
5 -682 819 -227 -585 711 599 457 390 282 731 -640 815 -446 -495
6 739 268 -47 -499 -830 -591 -650 -438 -236 210 -881 -146 494 838
7 579 286 529 210 -293 185 566 -189 732 46 -32 -675 425 -611
8 -436 412 -17 -262 -211 -801 216 -801 -532 614 -874 -283 -733 -892
9 -387 270 -236 796 255 61 280 -593 208 697 -301 710 140 -558
10 -477 338 863 -603 496 -896 568 45 505 -55 333 415 -886 -148
11 -920 591 432 -73 153 -975 -303 -174 647 -791 -990 956 724 453
12 412 -727 -25 398 -292 759 -318 129 246 660 -704 946 774 -234
13 -31 167 -759 -675 -76 595 -549 156 -847 256 -317 -584 240 -609
14 -913 141 -23 50 -185 -961 -43 -590 731 -90 911 43 -296 335
15 789 401 825 841 814 -209 291 -790 14 -526 -958 -192 56 -620
16 687 -98 848 -653 397 -510 -8 -827 944 536 -598 -815 440 0

```

Figure 2: arquivo de entrada do segundo caso de teste

Quinto caso: matriz com valores aleatórios de 0 a 9 de dimensão 10000x1000.

Sexto caso: matriz com valores aleatórios de -100 até 100 com dimensão igual a 25x25.

Sétimo caso: matriz com valores aleatórios de -10000 até 10000 com dimensão igual a 25x25.

Oitavo caso: matriz com valores aleatórios de -9 até 9 com dimensão igual a 25x25.

Nono caso: matriz com valores aleatórios de -100 até 100 com dimensão igual a 15x15.

Décimo caso: matriz com valores aleatórios de -100 até 100 com dimensão igual a 16x15.

Décimo primeiro caso: matriz com valores aleatórios de -100 até 100 com dimensão igual a 16x16.

Décimo segundo caso: matriz com valores aleatórios de -100 até 100 com dimensão igual a 25x7.

3.2 Resultados - Solução Dinâmica

Para testar a estratégia que utiliza a programação dinâmica, foram realizados testes com matrizes de diferentes tamanhos, algumas grandes demais para que a matriz seja escrita, que foram utilizadas apenas para verificar o tempo gasto pelo algoritmo.

Com o uso das funções para medir o tempo de execução e o uso de memória, a diferença entre os casos de teste se tornou evidente, principalmente para as matrizes maiores dos casos 3, 4 e 5. Já nas entradas 7, 8 e 9, foram usadas matrizes de dimensões iguais, porém a entrada 7 possuía inteiros que iam de -10000 até 10000, enquanto a oitava e a nona possuíam inteiros mais curtos, no entanto, não foi possível observar uma diferença significativa nos tempos de execução para esses três casos.

```

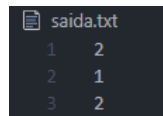
● iancrms@DESKTOP-4GF8G78:~/CCOMP/AEDS3/TP2$ ./tp2 1 -i entrada1.txt
Tempo de execucao: 0.000221 segundos
Uso de memoria: 2452 kilobytes
● iancrms@DESKTOP-4GF8G78:~/CCOMP/AEDS3/TP2$ ./tp2 1 -i entrada2.txt
Tempo de execucao: 0.000172 segundos
Uso de memoria: 2452 kilobytes
● iancrms@DESKTOP-4GF8G78:~/CCOMP/AEDS3/TP2$ ./tp2 1 -i entrada3.txt
Tempo de execucao: 0.004627 segundos
Uso de memoria: 2452 kilobytes
● iancrms@DESKTOP-4GF8G78:~/CCOMP/AEDS3/TP2$ ./tp2 1 -i entrada4.txt
Tempo de execucao: 0.411536 segundos
Uso de memoria: 74980 kilobytes
● iancrms@DESKTOP-4GF8G78:~/CCOMP/AEDS3/TP2$ ./tp2 1 -i entrada5.txt
Tempo de execucao: 0.523381 segundos
Uso de memoria: 119328 kilobytes
● iancrms@DESKTOP-4GF8G78:~/CCOMP/AEDS3/TP2$ ./tp2 1 -i entrada6.txt
Tempo de execucao: 0.000361 segundos
Uso de memoria: 2452 kilobytes
● iancrms@DESKTOP-4GF8G78:~/CCOMP/AEDS3/TP2$ ./tp2 1 -i entrada7.txt
Tempo de execucao: 0.000243 segundos
Uso de memoria: 2452 kilobytes
● iancrms@DESKTOP-4GF8G78:~/CCOMP/AEDS3/TP2$ ./tp2 1 -i entrada8.txt
Tempo de execucao: 0.000245 segundos
Uso de memoria: 2456 kilobytes
● iancrms@DESKTOP-4GF8G78:~/CCOMP/AEDS3/TP2$ ./tp2 1 -i entrada9.txt
Tempo de execucao: 0.000256 segundos
Uso de memoria: 2456 kilobytes

```

Figure 3: tempo de execução e uso de memória para os casos 1 - 9

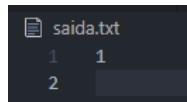
Em relação ao valor de energia mínima da primeira casa, ou seja, a solução do problema, não foi possível confirmá-lo nas matrizes maiores, mas para matrizes menores, como nos casos de teste 1 e 2, as soluções encontradas foram ótimas. Além disso, um fato curioso que pôde ser observado foi que ao testar em matrizes com elementos que variavam entre $-n$ e n , ou seja, de n números antes de 0 até n números depois de 0, quando a primeira casa possuía pelo menos um vizinho com valor positivo, na maioria das matrizes existia um caminho que tinha como energia inicial mínima 1, ou seja, toda energia que ele perdia no caminho ele conseguia encontrar antes para acrescentar ao seu saldo.

Saída do primeiro e do segundo caso de teste:



```
saida.txt
1 2
2 1
3 2
```

Figure 4: arquivo de saída do primeiro caso de teste



```
saida.txt
1 1
2 [REDACTED]
```

Figure 5: arquivo de saída do segundo caso de teste

Apesar de não saber se a solução imprimida no arquivo de saída era a solução ótima para matrizes muito grandes, foram feitos testes como por exemplo printar as matrizes de energia mínima e de saldo de energia e também de alterar os valores dos vizinhos da posição 1 para negativos para ver se o resultado mudava e se as matrizes mudavam também, e o código aparentou funcionar normalmente.

3.3 Resultados - Solução Recursiva

Foram realizados alguns testes para verificar o funcionamento da estratégia recursivo, dentro de suas limitações. O primeiro teste foi realizado com as três matrizes disponibilizadas na apresentação do problema (primeiro caso de teste).

Na primeira matriz, começando pela posição (0,0), o algoritmo recursivo analisa os valores armazenados nas posições (1,0) e (0,1). A partir daí, ele faz uma verificação para descobrir qual o menor valor entre eles e determinar o melhor caminho a ser seguido. Nesse caso, Harry pode avançar para qualquer um dos dois caminhos, visto que eles armazenam o mesmo valor. O algoritmo fará isso recursivamente para as próximas posições. No próximo passo, Harry avança para a posição (2,2), onde irá ocorrer a perda de 2 unidades de vida. Por fim, ele atinge a posição (2,3) e alcança o artefato mágico. É preciso que ele comece com pelo menos 2 de vida, para que sua vida não chegue à 0 em nenhum momento do trajeto.

Na segunda matriz do teste, o caminho escolhido é: (0,0) → (1,0) → (1,1). Como não há nenhum valor negativo nessa matriz, Harry não perde vida em nenhum momento, portanto, é preciso que ele tenha apenas 1 de vida para começar e completar o trajeto sem que sua vida atinja 0 durante o percurso.

Já na terceira matriz, o caminho tomado é: (0,0) → (1,0) → (1,1) → (1,2) → (1,3) → (2,3). Para alcançar o artefato, é preciso que Harry inicie com pelo menos 2 de vida, já que haverá a subtração de 1 de vida na posição (1,0) da matriz.

Após isso, o algoritmo foi testado para o segundo caso de teste, demorando, em média, 0.2 segundos, e retornando a solução ótima do problema que é 1.

Ainda, um outro teste foi conduzido com uma matriz que possui 25 linhas e 7 colunas, a fim de verificar a eficiência do algoritmo para matrizes maiores. Com esse teste, foi possível definir um limite para o tamanho máximo da matriz para que a solução mantenha sua eficiência. Dito isso, a estratégia recursiva é tão eficiente quanto à dinâmica para matrizes cuja soma de linhas e colunas não ultrapasse 32.

3.4 Comparação entre os 2 algoritmos

Em relação à diferença entre as duas estratégias, foi possível visualizar a diferença entre a complexidade nos tempos de execução para os casos com matrizes menores em que a estratégia recursiva conseguiu executar em menos de 5 segundos.

```
iancrms@DESKTOP-4GF8G78:~/CCOMP/AEDS3/TP2$ ./tp2 1 -i entrada1.txt
Tempo de execucao: 0.000433 segundos
Uso de memoria: 2456 kilobytes
iancrms@DESKTOP-4GF8G78:~/CCOMP/AEDS3/TP2$ ./tp2 1 -i entrada2.txt
Tempo de execucao: 0.000170 segundos
Uso de memoria: 2456 kilobytes
iancrms@DESKTOP-4GF8G78:~/CCOMP/AEDS3/TP2$ ./tp2 1 -i entrada9.txt
Tempo de execucao: 0.000360 segundos
Uso de memoria: 2456 kilobytes
iancrms@DESKTOP-4GF8G78:~/CCOMP/AEDS3/TP2$ ./tp2 1 -i entrada10.txt
Tempo de execucao: 0.000563 segundos
Uso de memoria: 2456 kilobytes
iancrms@DESKTOP-4GF8G78:~/CCOMP/AEDS3/TP2$ ./tp2 1 -i entrada11.txt
Tempo de execucao: 0.000274 segundos
Uso de memoria: 2456 kilobytes
iancrms@DESKTOP-4GF8G78:~/CCOMP/AEDS3/TP2$ ./tp2 1 -i entrada12.txt
Tempo de execucao: 0.000160 segundos
Uso de memoria: 2456 kilobytes
```

Figure 6: tempo de execução dos casos de teste 1, 2, 9, 10, 11 e 12 utilizando a estratégia 1

```

● ianrms@DESKTOP-4GF8G78:~/CCOMP/AEDS3/TP2$ ./tp2 2 -i entrada1.txt
Tempo de execucao: 0.000200 segundos
Uso de memoria: 2456 kilobytes
● ianrms@DESKTOP-4GF8G78:~/CCOMP/AEDS3/TP2$ ./tp2 2 -i entrada2.txt
Tempo de execucao: 0.209016 segundos
Uso de memoria: 2456 kilobytes
● ianrms@DESKTOP-4GF8G78:~/CCOMP/AEDS3/TP2$ ./tp2 2 -i entrada9.txt
Tempo de execucao: 0.809334 segundos
Uso de memoria: 2456 kilobytes
● ianrms@DESKTOP-4GF8G78:~/CCOMP/AEDS3/TP2$ ./tp2 2 -i entrada10.txt
Tempo de execucao: 1.521384 segundos
Uso de memoria: 2456 kilobytes
● ianrms@DESKTOP-4GF8G78:~/CCOMP/AEDS3/TP2$ ./tp2 2 -i entrada11.txt
Tempo de execucao: 3.113365 segundos
Uso de memoria: 2456 kilobytes
● ianrms@DESKTOP-4GF8G78:~/CCOMP/AEDS3/TP2$ ./tp2 2 -i entrada12.txt
Tempo de execucao: 0.024523 segundos
Uso de memoria: 2456 kilobytes

```

Figure 7: tempo de execução dos casos de teste 1, 2, 9, 10, 11 e 12 utilizando a estratégia 2

4 Conclusão

Em conclusão, essa documentação apresenta duas soluções para o problema de determinar a energia mínima necessária para o Harry Potter alcançar um artefato mágico em um grid. Além disso, destaca a importância de problemas de otimização de caminho em diversas áreas, desde jogos até a engenharia, uma vez que as soluções apresentadas para esse problema em específico podem ser reaproveitadas para solucionar problemas de mesma natureza por outros desenvolvedores.

Como foram utilizadas 2 soluções com complexidades diferentes, para testes em matrizes maiores essa diferença foi notável nos tempos de execução, e para os testes com matrizes menores, em que era possível visualizar a solução na matriz principal, os resultados foram confirmados como ótimos.

Esse problema, portanto, foi bastante proveitoso em relação ao desenvolvimento de nossas habilidades em encontrar soluções através da criação de um algoritmo, e os 2 algoritmos criados foram um bom exemplo de como a solução pode ser encontrada por mais de 1 método, com complexidades diferentes.